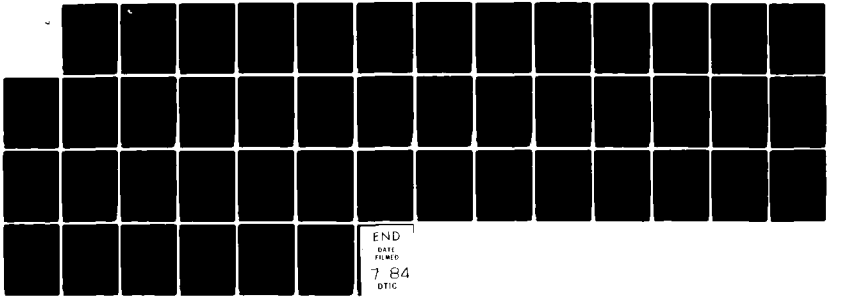
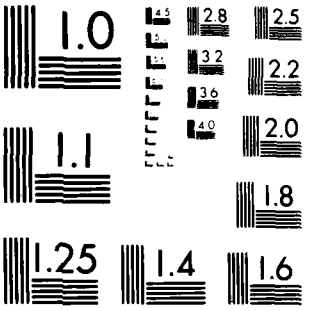
AD-A142 010 CANONICAL REALIZATIONS OF COMPLETELY REGULAR MODULAR COMPUTING NETWORKS(U) INTEGRATED SYSTEMS INC PALO ALTO CA H LEV-ARI MAY 84 ISI-41 N00014-83-C-0377 1/1

UNCLASSIFIED F/G 9/2 NL



END
DATE FILMED
7 84
DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

integrated systems inc.

(12)

CANONICAL REALIZATIONS OF COMPLETELY REGULAR MODULAR COMPUTING NETWORKS

HANOCH LEV-ARI

AD-A142 010

PREPARED FOR:

OFFICE OF NAVAL RESEARCH
800 NORTH QUINCY STREET
ARLINGTON, VIRGINIA 22217

ATTENTION: DR. DAVID W. MIZELL

PREPARED UNDER:
CONTRACT NO. N00014-83-C-0377

DTIC ELECTE
JUN 1 3 1984
A

DTIC FILE COPY

This document has been approved for public release and sale; its distribution is unlimited.

ISI REPORT 41 • MAY 1984

151 University Avenue Palo Alto, CA. 94301 Ph. 415/321-9773

84 05 23 011

TABLE OF CONTENTS

A-1

SECTION 1
INTRODUCTION

The multiplicity of possible hardware implementations for a given computational scheme is efficiently displayed by a space-time representation, a notational tool that has been incorporated into some recent methodologies for modeling, analysis and design of parallel architectures [1-9]. Coordinate transformations of a given space-time representation produce distinct hardware configurations which are equivalent in the sense of being the implementations of the same computational scheme. The problem of mapping a given algorithm into a desired hardware configuration can, therefore, be partly reduced to choosing the appropriate coordinate transformation in space-time. In particular, uniform recurrence relations, which correspond to systolic-array architectures, are described by regular space-time representations. This implies that only linear coordinate transformations are required, and that the entire computational scheme can be described by a small collection of vectors in space-time, the dependence vectors [3,5,6,8]. Consequently, the selection of a desired hardware architecture for a given algorithm reduces to the determination of an appropriate nonsingular matrix with integer entries.

Previous research has focused upon the algebra of such transformation matrices in multidimensional linear spaces, establishing conditions for the mappability of given algorithms into systolic-array architectures. However, since physical space is 3-dimensional the dimensionality of space-time cannot exceed 4. Moreover, since VLSI hardware must be planar and no intersection of wires is allowed, only 3-dimensional space-time representations are allowed. Consequently, the number of distinct systolic-array architectures is very small; Miranker and Winkler have, in fact, shown that only three topologies--linear, rectangular, hexagonal--are permitted for systolic arrays.

This paper establishes a simple technique for transforming a given 3-dimensional space-time representation into an equivalent canonical form. A

catalogue of canonical forms is constructed, showing a total of 34 distinct systolic architectures. The task of selecting an appropriate transformation for a given space-time representation reduces, therefore, to the determination of the equivalent canonical form. The important result, which has been overlooked in previous research, is that the canonical equivalent of any given space-time representation is unique. This means that once a space-time representation has been specified there is no flexibility left in the process of mapping into systolic-array architectures.

A small fraction of space-time representation do allow some flexibility in selecting the hardware architecture, but only at the cost of inefficient implementation. The well-known example of matrix multiplication, which has four distinct realizations (see [11]-[14]) turns out to be one of the few cases where such flexibility is available. A closer examination of the structure of the matrices to be multiplied reveals that each realization is efficient under a different set of structural assumptions (see Section 4.3). Thus, in summary, carefully specified algorithms lead to unique space-time representations which, in turn, lead to essentially unique architectures.

SECTION 2

COMPLETELY REGULAR MCNs

A modular computing network (MCN) can be loosely defined as an association of multivariable functions with the vertices of a directed acyclic graph (see [7] for a rigorous definition). More precisely, multivariable functions are associated only with internal vertices, which are those vertices that have both incoming and outgoing arcs. A vertex with ρ_i incoming arcs and ρ_o outgoing arcs is associated with an input-output map with ρ_i input variables and ρ_o output variables.

A completely regular MCN is one that can be represented by a regular multidimensional grid, and in which all input-output maps associated with the vertices are identical. Thus, the vertices of a completely regular MCN can be mapped into points of the multidimensional grid Z^n in the n-dimensional Euclidean space R^n, where Z denotes the set of integers; the arcs of a completely regular MCN become vectors (n-tuples of real numbers) representing the directed straight lines connecting points of the grid Z^n. Clearly, not all points in Z^n correspond to vertices of the MCN. Those that do determine the <u>domain</u> Γ of the MCN in Z^n. The requirement of complete regularity translates into the statement that the vectors (arcs) emanating from any point (vertex) in Γ do not depend upon the choice of vertex. Consequently, the entire MCN is characterized by:

(i) the set of <u>dependence vectors</u> $\{d_i\}$ emanating from a single vertex;

(ii) the domain $\Gamma \subset Z^n$;

and

(iii) the input output map f

$$f: (x_1, \ldots, x_p) \dashrightarrow (y_1, \ldots, y_p)$$

associated with every vertex in the domain Γ.

A curious consequence of this definition is that the input-output map f has the same number of inputs and outputs, since the number of arcs emanating from a point in Γ is always the same as the number of arcs converging to a point.

Not every set of dependence vectors $\{d_i\}$ determines a valid MCN. For instance, the directed graph representing an MCN has to be acyclic. In terms of dependence vectors this means that it is impossible to find positive integers $\{k_i\}$ such that $\sum k_i d_i = 0$. Another requirement is that the ancestry of every vertex $v \in \Gamma$ (i.e., the set of all points from which v can be reached) has to be finite. This constraint is trivial if Γ is a finite set; however, if Γ is infinite (as is often the case with signal processing algorithms) this constraint implies that Γ has to be bounded in the directions $\{-d_i\}$.

In the sequel we shall focus upon completely regular MCNs in Z^3, because such MCNs correspond to space-time representation of planar systolic-array-like architectures (see [3] - [7]). We shall impose the constraint of causality resulting from the association of 'time' with one of the coordinate axes in Z^3 and examine the flow of data through the architecture in terms of the dependence vectors characterizing the MCN.

2.1 SPACE-TIME REPRESENTATIONS IN Z^3

MCNs in Z^3 are characterized by 3-dimensional dependence vectors $\{d_i\}$, which we shall represent by row vectors of length 3. The collection of all dependence vectors

$$D := [d_i]_{i=1}^{p} \tag{2.1}$$

forms a $p \times n$ matrix, which we shall call the dependence matrix. The boundary of the domain Γ can always be described as a polyhedron. It will be sufficient for our purposes to consider only convex polyhedra, and in fact, only those that can be characterized in terms of the dependence vectors (see Section 3.4 for a further discussion of this choice).

The interpretation of MCNs in Z^3 as space-time representations of hardware architectures imposes the additional constraint of causality: every dependence vector must have a positive time coordinate, since computation and propagation of data cannot be accomplished in zero time. Moreover, since data cannot propagate faster than the speed of electromagnetic waves in metallic conductors, the directions of dependence vectors must lie within a certain cone, the time-like cone in the space-time continuum. By appropriate scaling of space and time coordinates we can reduce this condition to the requirement

$$d_i \, [0\ 0\ 1]^* \geq 1 \qquad (2.2)$$

which means that the third coordinate of d_i must be (an integer) larger or equal to 1.

The association of time with the third coordinate of dependence vectors allows us to express the finite ancestry condition in simple form. The exclusion of ancestors that are infinitely remote from a given vertex in the domain Γ is equivalent to the requirement that Γ be a subspace of the positive half space of Z^3, i.e., the half space corresponding to non-negative time coordinates. Moreover, since hardware must always be finite, the spatial extent of Γ must be bounded. Thus, the only direction in which Γ may remain unbounded is that of positive time, corresponding to a computation that continues indefinitely in time (e.g., a filtering of an infinite time-series), but produces results (outputs) at regular intervals.

Vertices in Γ that share the same spatial coordinates are considered as representing the same hardware processor at different instances in time. Regularity implies that such isospatial vertices are spread in time at regular intervals. This interval, which is the same for all processors, will be called the periodicity index of the architecture. The periodicity index corresponding to a given dependence matrix D is the smallest solution π of the equation

$$\eta D = \pi [0 \quad 0 \quad 1] \tag{2.3}$$

where η is any row vector with integer (possibly negative) entries. To prove this result we notice that ηD is an integer combination of dependence vectors; moreover, if $v(x_1,y_1,t_1)$ and $v(x_2,y_2,t_2)$ are two distinct vertices in Γ, then the vector connecting these vertices can always be expressed in the form ηD for an appropriate (possibly nonunique) row vector η. If the two vertices share the same spatial coordinates, then their interconnecting vector is colinear with $[0 \quad 0 \quad 1]$, and so (2.3) is satisfied for some η,π. Finally, the smallest temporal displacement is obtained when π is minimized in (2.3). The periodicity index π can, actually, be evaluated without an exhaustive search through all possible integer vectors of η that satisfy (2.3), as is demonstrated in Section 2.2.

The most important attribute of the space-time representation of a completely regular MCN is the invariance of the MCN under coordinate transformations in space-time. This is so because coordinate transformations do not affect the interconnection pattern of the space-time representation, and consequently leave the corresponding directed graph unaltered. In the case of regular space-time representations it is sufficient to consider the effect of linear coordinate transformations; this is done in detail in Sections 3 and 4.

2.2 SPATIAL PROJECTION OF MCNs IN Z^3

The first two coordinates in a three-dimensional space-time can be interpreted as physical space. When a space-time representation is projected into the plane formed by the first two coordinates, vertices represent computing agents (i.e., processors) and arcs represent physical interconnections (i.e., wires). The projection amounts to the truncation of each dependence vector to its first two coordinates, viz.,

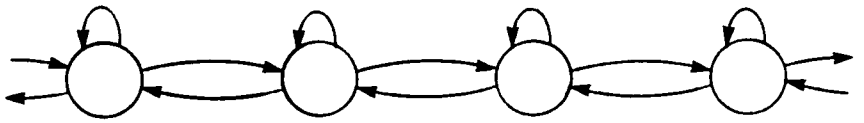
$$D_s := D \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \tag{2.4}$$

The truncated dependence matrix D_s ('s' stands for 'spatial') is usually sufficient to characterize the architecture, since we commonly assume that each dependence vector represents a computation that requires a unit of time, and consequently

$$D = \begin{bmatrix} & & 1 \\ & D_s & 1 \\ & & \vdots \\ & & 1 \end{bmatrix} \tag{2.5}$$

This assumption is violated only when D has a periodicity index $\pi(D) > 1$ and, in addition, D contains a dependence vector of the form $[0 \;\; 0 \;\; \tau]$. This dependence vector is truncated to $[0 \;\; 0]$, so τ cannot be recovered unless $\tau = \pi$ or $\tau = 1$. These, in fact, are the only two possible values for τ as explained in Section 4.4.

The truncated dependence matrix can be pictorially represented by a conventional block-diagram such as Figure 2-1. Each truncated dependence vector is represented by an arc with the appropriate spatial displacement, while truncated dependence vectors of the form $[0 \;\; 0]$, which correspond to local memory, are represented by self-arcs.



a. Block-Diagram Representation

$$D = \begin{bmatrix} 1 & 0 & 1 \\ -1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \qquad D_s = \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 0 \end{bmatrix}$$

b. Dependence and Truncated Dependence Matrices

Figure 2-1. Example of a Regular Hardware Architecture

The truncated equivalent of (2.3) becomes

$$\eta D_s = 0 \tag{2.6}$$

so that every feasible choice of η corresponds to an undirected loop in the 2-dimensional block-diagram representation. Thus, every feasible η is obtained by considering all possible loops in the block-diagram representation. If there are no self-loops on vertices, then D_s contains no zero row and (2.5) holds. Consequently, by (2.3),

$$\eta D[0 \quad 0 \quad 1]^* = \eta[1 \quad 1 \quad . \quad . \quad . \quad 1]^* = \pi$$

so π is obtained by adding up the entries of η. This is, in fact, done by counting each arc along the loop as 1 if it coincides with the orientation of the loop and as -1 if it points in the reverse direction. Since the smallest value of π is required, only the shortest loops need to be considered. We shall show in Section 3.3 that π never exceeds 3 and is seldom larger than 1.

SECTION 3

CLASSIFICATION OF HARDWARE ARCHITECTURES

Completely regular MCNs were characterized in the previous section in terms of their dependence vectors. It was also indicated that MCNs with different dependence vectors may nevertheless be equivalent, namely they will have equivalent space-time representations. The equivalence of completely regular MCNs is easy to verify, since it amounts to the existence of a nonsingular linear transformation relating the dependence matrices of the MCNs in consideration.

The study of equivalence can be carried out at several different levels of abstraction. At the lowest (most detailed) level each completely regular MCN is represented by a dependence matrix

$$D := [d_i]_{i=1}^{p} \tag{3.1}$$

where d_i are row vectors of length 3 whose first two coordinates represent the planar space of integrated circuits and the third coordinate represents time. Thus, for instance, the MCN of Figure 3-1 is characterized by the dependence matrix

$$D = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Notice that the time coordinate of all three dependence vectors equals to 1, reflecting the assumption that each dependence vector represents a computation that requires a unit of time. This assumption can, of course, be modified to incorporate computations with unequal processing times. Notice also that the direction of dependence vectors coincides with that of the arrows in Figure 3-1, pointing toward the successors of a given processor, rather than toward the predecessors of the same processor, as in [3].

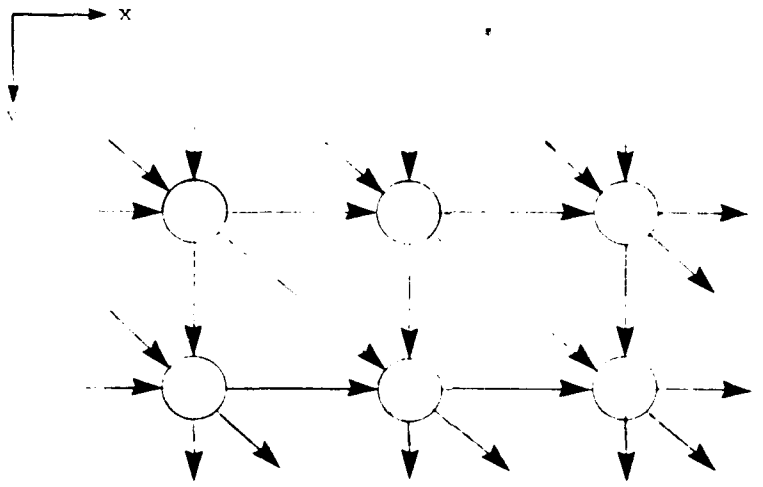


Figure 3-1. Example of a Completely Regular MCN

At the intermediate level of abstraction only the spatial coordinates of each dependence vector are considered. This results in the elimination of the third column of the dependence matrix D, resulting in the truncated dependence matrix D_s

$$D_s = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$$

for the example of Figure 3-1. We shall show in the following section that the truncated dependence matrix D_s provides, in fact, a complete, albeit implicit, characterization of the MCN. This characterization can be transformed in a unique manner into the explicit characterization D.

At the highest level of abstraction only the topology of the hardware is considered. This means that the directed graph representing the flow of data is replaced by the corresponding non-directed graph. Thus, for instance, the MCN of Figure 3-1 and that of Figure 3-2 are topologically equivalent, even though the latter has a different dependence matrix, viz.

$$D_s = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -1 & -1 \end{bmatrix}$$

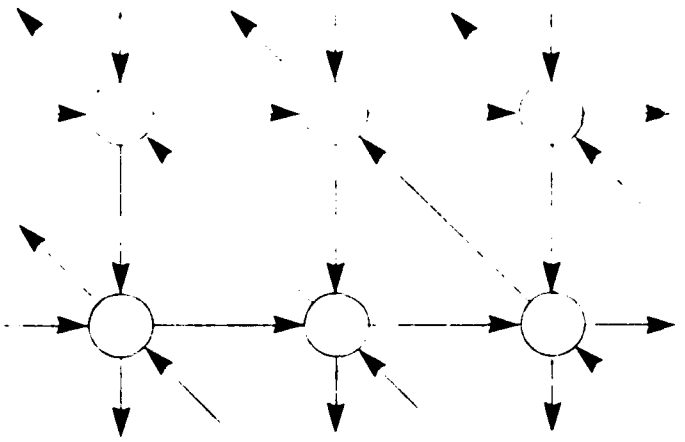


Figure 3-2. A Completely Regular MCN Which is Topologically Equivalent to that of Figure 3-1

This section is devoted to the study of topological equivalence followed by the study of architectural (D_s) equivalence. The more complicated topic of space-time equivalence is presented in the following section, where it is also shown that distinct hardware configurations may, nevertheless, have equivalent space-time representations.

3.1 TOPOLOGICAL EQUIVALENCE

The topic of topological equivalence has been studied by Miranker and Winkler [3], who have shown that there are only three distinct topologies (Figure 3-3):

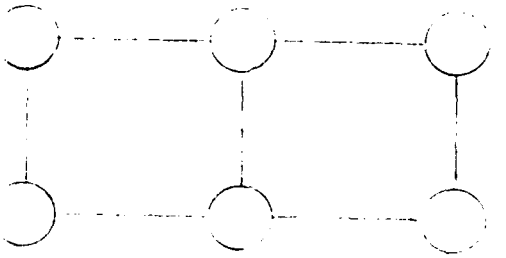
(1) The linear topology, with a single dependence vector,

$$D_s = [1 \quad 0]$$

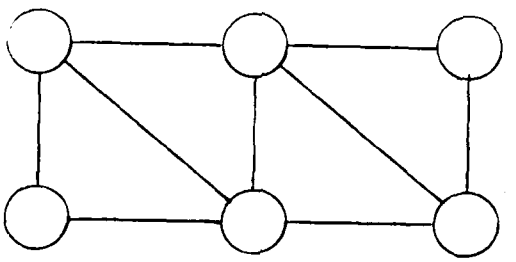
(2) The rectangular topology, with two dependence vectors,

$$D_s = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

a. The Linear Topology



b. The Rectangular Topology



c. The Hexagonal Topology

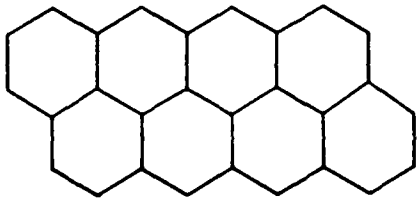
Figure 3-3. The Three Fundamental Topologies

(3) The hexagonal topology, with three dependence vectors,

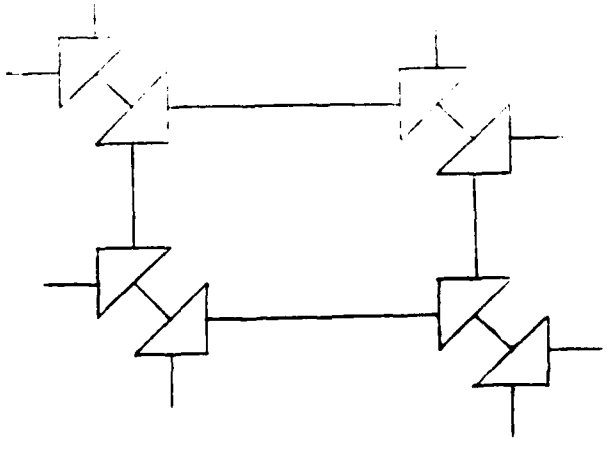
$$D_s = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$$

Every systolic-array-like architecture can be related by a linear transformation to one of these fundamental topologies. Also, it is impossible to have more than three non-colinear dependence vectors in a planar architecture.

The same conclusion can be reached by a graph-theoretic argument. The graph describing the hardware configuration of a completely regular MCN is clearly a mosaic, i.e., a planar graph in which all faces are bounded the same number of edges and all vertices (except those on the external boundary of the graph) have the same number of incident edges. As is well known, there are only three possible mosaics [15]: triangular, rectangular and hexagonal. The triangular mosaic has vertices of degree 6 and coincides with the hexagonal topology. The rectangular mosaic has vertices of degree 4 and coincides with the rectangular topology. The hexagonal mosaic (Figure 3-4) does not correspond to a completely regular MCN, since it requires two sets of dependence vectors rather than one. However, it can be rearranged by combining pairs of adjacent processors into a single processor (Figure 3-4b), so that the resulting configuration has a rectangular topology. Thus, there are only two mosaics corresponding to completely regular MCNs, which combined with the linear configuration makes a total of 3 fundamental topologies.



a. The Mosaic



b. Rearrangement as a Rectangular Topology

Figure 3-4. The Hexagonal Mosaic

3.2 ARCHITECTURAL EQUIVALENCE

Each of the interconnecting wires in the three fundamental topologies can be associated with two direction vectors, one pointing along the wire in one way, the other in the reverse. This makes a total of three possibilities for each interconnecting wire: (i) + d, (ii) -d, and (iii) $\pm$ d. This means that the linear topology results in $3^1 = 3$ architectures, the rectangular topology in $3^2 = 9$ architectures and the hexagonal topology in $3^3 = 27$ architectures. Since many of these architectures are equivalent, a classification of the distinct architectures is provided in Table 3-1. The nomenclature consists of a capital letter (L, R or H) indicating the topology (linear, rectangular or hexagonal), a digit indicating the number of dependence vectors and a lower case letter, whenever required, to distinguish between architectures which have the same topology and the same number of dependence vectors but are not equivalent, e.g., H3a and H3b. The table lists all equivalent configurations in a single row.

3.3 PERIODICITY ANALYSIS AND THROUGHPUT

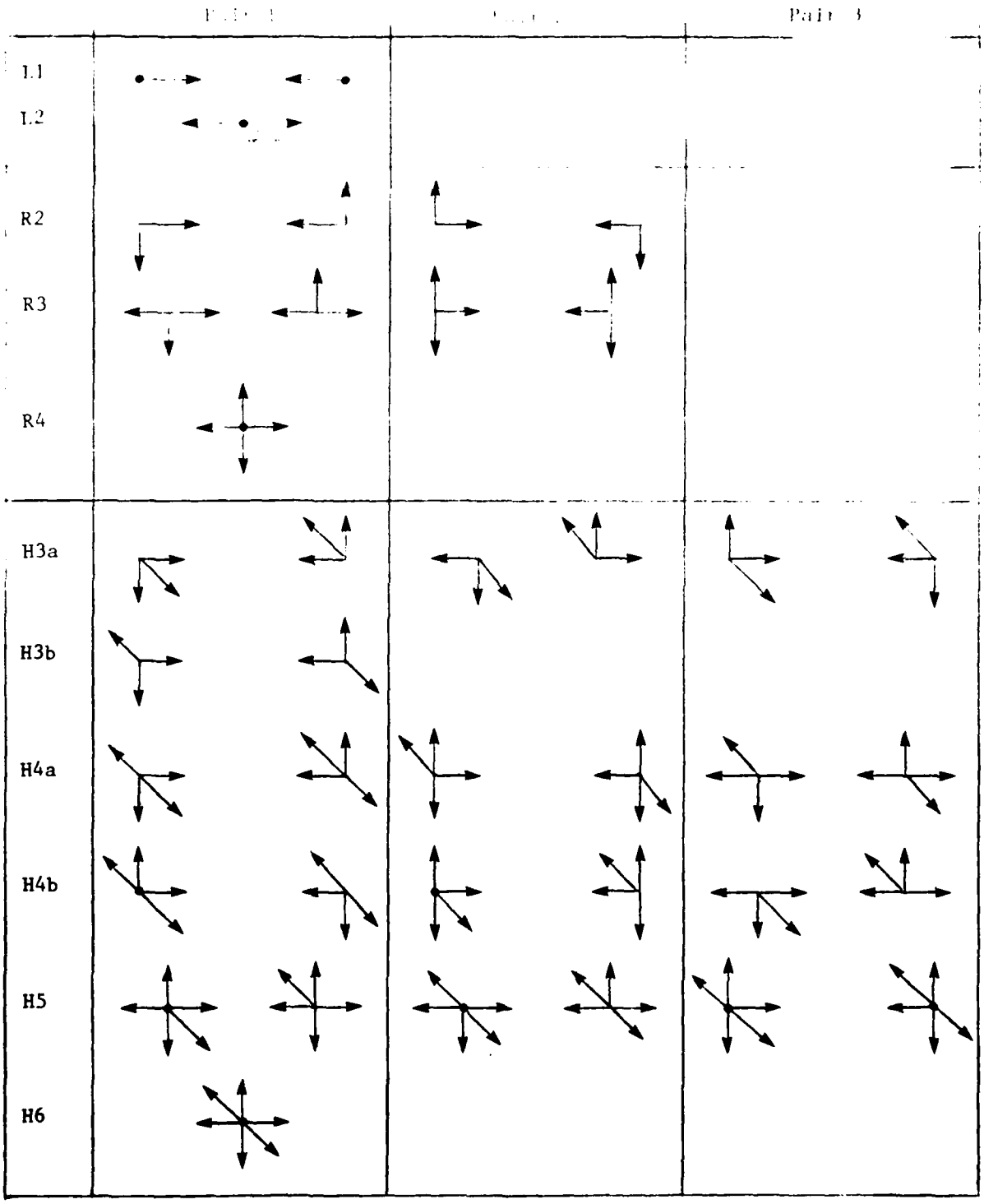
The occurrence of cycles (i.e., closed loops of directed arcs) in the directed graph representing a hardware architecture provides important information about the throughput rate of the architecture. In this subsection we analyze this information and identify the configurations with low throughput.

The periodicity index π of architectures has been defined in Section 2.2. It can be computed either by examining undirected loops in the graph representing the architecture or by solving the equation

$$\eta D_s = 0 \qquad (3.2)$$

for every possible row vector η with integer elements, and summing the elements of η. The periodicity index π equals the smallest of these

TABLE 3-1. CLASSIFICATION OF HARDWARE ARCHITECTURES



sums. If no solution η exists, π is defined to be 1. Following this technique we conclude that L1, R2 have no solution and have a unit periodicity index, while other architectures have solutions, as follows:

(i) L2 has $\eta = [1\ \ 1]$; hence $\pi = 2$.

(ii) R3 has $\eta = [1\ 1\ 0]$; hence $\pi = 2$.

(iii) R4 has $\eta = [1\ 1\ 0\ 0], [0\ 0\ 1\ 1]$; hence $\pi = 2$.

(iv) H3a has $\eta = [1\ 1\ -1]$; hence $\pi = 1$.

(v) H3b has $\eta = [1\ 1\ 1]$; hence $\pi = 3$.

(vi) H4a has $\eta = [1\ 1\ -1\ 0], [0\ 0\ 1\ 1], [1\ 1\ 0\ 1]$; hence $\pi = 1$.

(vii) H4b has $\eta = [1\ -1\ 0\ 1], [0\ 0\ 1\ 1]$; hence $\pi = 1$.

(viii) H5 has $\eta = [1\ 0\ 1\ 0\ -1], [1\ 1\ 0\ 0\ 0], [0\ 0\ 1\ 1\ 0]$; hence $\pi = 1$.

(ix) H6 has $\eta = [1\ 0\ 1\ 0\ -1\ 0], [1\ 1\ 0\ 0\ 0\ 0], [0\ 0\ 1\ 1\ 0\ 0], [0\ 0\ 0\ 0\ 1\ 1]$; hence $\pi = 1$.

In the sequel we shall measure the throughputs of architectures relative to the throughput of the linear architecture L1 (a classical pipeline). Since the time interval between two successive applications of input equals the periodicity index, the relative throughput of a given architecture is given by the formula

$$\text{relative throughput} = \frac{1}{\text{periodicity index}} \qquad (3.3)$$

Thus, the relative throughput of L2, R3, R4 is 1/2 and that of H3b is 1/3.

3.4 BOUNDARY ANALYSIS

No assumption has been made up to this point about the shape of the boundary of a given hardware architecture. However, since the shape of the boundary is changed by linear transformation it has to be taken into consideration in the process of classifying architectures. As an example consider the 6 equivalent configurations denoted by H3a (Table 3-1). The truncated dependence matrices of the first and third of these configurations are related by a linear transformation, viz.

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \\ -1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ -1 & 0 \end{bmatrix}$$

Now assume that the first configuration has a rectangular boundary, which can be characterized by boundary matrix

$$B_s = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

consisting of all dependence vectors colinear with the boundary. The linear transformation maps this boundary into

$$B_s' = B_s \begin{bmatrix} 1 & 1 \\ -1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ -1 & 0 \end{bmatrix}$$

which characterizes a parallelogram rather than a rectangle. Thus, the first H3a configuration with a rectangular boundary is equivalent to the third H3a configuration with a parallelogram boundary. It is not equivalent, however, to the third H3a configuration with a rectangular boundary. Clearly, we need to reclassify the entries of Table 3-1 according to both the dependence matrix and the boundary.

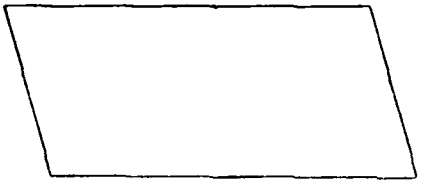
We shall be concerned only with boundaries that satisfy the two following conditions:

(i) The boundary curve is a closed convex polygon

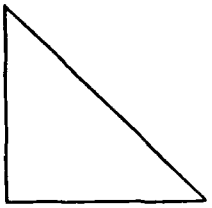
(ii) Each segment of the boundary curve is colinear with some dependence vector.

Thus, the only possible directions for the segments of the boundary curve are [1 0], [0 1] and [1 1]. Consequently, there are four possible boundary curves (Figure 3-5): rectangle, parallelogram, lower triangle, upper triangle. Of these, only the rectangle-shape boundary can be applied to the linear (1) and rectangular (R) architectures. On the other hand, all four possible boundaries can be combined with hexagonal (H) architectures. However, since linear transformations map rectangles into parallelograms and lower triangles into upper ones, we need only consider the combination of each hexagonal entry of Table 3-1 with either a rectangular or a triangular boundary.

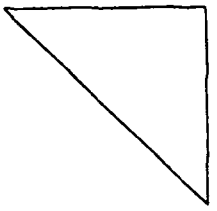
a. Rectangle



b. Parallelogram



c. Lower Triangle



d. Upper Triangle

Figure 3-5. Fundamental Boundary Curves

TABLE 3-2. CLASSIFICATION OF HARDWARE ARCHITECTURES WITH RECTANGULAR BOUNDARIES

L1	L2	R2	R3	R4
1 0	1 0	1 0	1 0	1 0
	-1 0	0 1	-1 0	-1 0
			0 1	0 1
				0 -1

H3aα	H3aβ	H3b	H4aα	H4aβ
1 0	-1 0	1 0	1 0	1 0
0 1	0 1	0 1	0 1	0 1
1 1	1 1	-1 -1	1 1	-1 0
			-1 -1	-1 -1

H4bα	H4bβ	H5α	H5β	H6
1 0	1 0	1 0	1 0	1 0
0 -1	0 1	-1 0	-1 0	-1 0
1 1	0 -1	0 1	0 1	0 1
-1 -1	1 1	0 -1	1 1	1 1
		1 1	-1 -1	-1 -1

With rectangular boundaries we need to consider matrices of the form

$$\begin{bmatrix} D_s \\ B_s \end{bmatrix} \quad \begin{bmatrix} D_s \\ I \end{bmatrix} \tag{3.4}$$

Clearly

$$\begin{bmatrix} D_s \\ I \end{bmatrix} \cdot (-1) \cdot \begin{bmatrix} -D_s \\ -I \end{bmatrix} \sim \begin{bmatrix} -D_s \\ I \end{bmatrix}$$

which shows that the reversal of all dependence vectors does not produce a new configuration. The 6 entries in each one of the rows H3a, H4a, H4b, H5 of Table 3-1 can, therefore, be considered as 3 pairs of conjugate configurations. Of these, the second and third pair are still equivalent when combined with rectangular boundaries, but the first pair is different. Thus, the entries of Table 3-1, when combined with rectangular boundaries, can be reclassified as in Table 3-2. This time each architecture is specified by its D_s matrix rather than by a pictorial description as in Table 3-1.

Similarly, we can combine each hexagonal entry of Table 3-1 with a lower triangular boundary. This will again produce two distinct architectures for each one of the rows H3a, H4a, H4b, H5. However, there is no need to do it explicitly, since the resulting configurations can always be obtained by 'cutting' the appropriate hexagonal topology combined with a rectangular boundary along the main diagonal. Thus, it will be sufficient to focus in the sequel upon rectangular boundaries alone.

3.5 SUMMARY

Systolic-array-like architectures have been classified by topology, interconnection pattern and shape of boundary. We have shown that there are only 15 distinct (non-equivalent) architectures (see table 3-2). We have also shown that it is sufficient to consider only rectangular boundaries which are of practical importance in the process of VLSI layout.

A genealogical chart (Figure 3-6) shows which architectures are contained in any given architecture. In particular, it shows that H6 is the 'universal architecture' for systolic arrays, containing every possible architecture with a smaller number of dependence vectors.

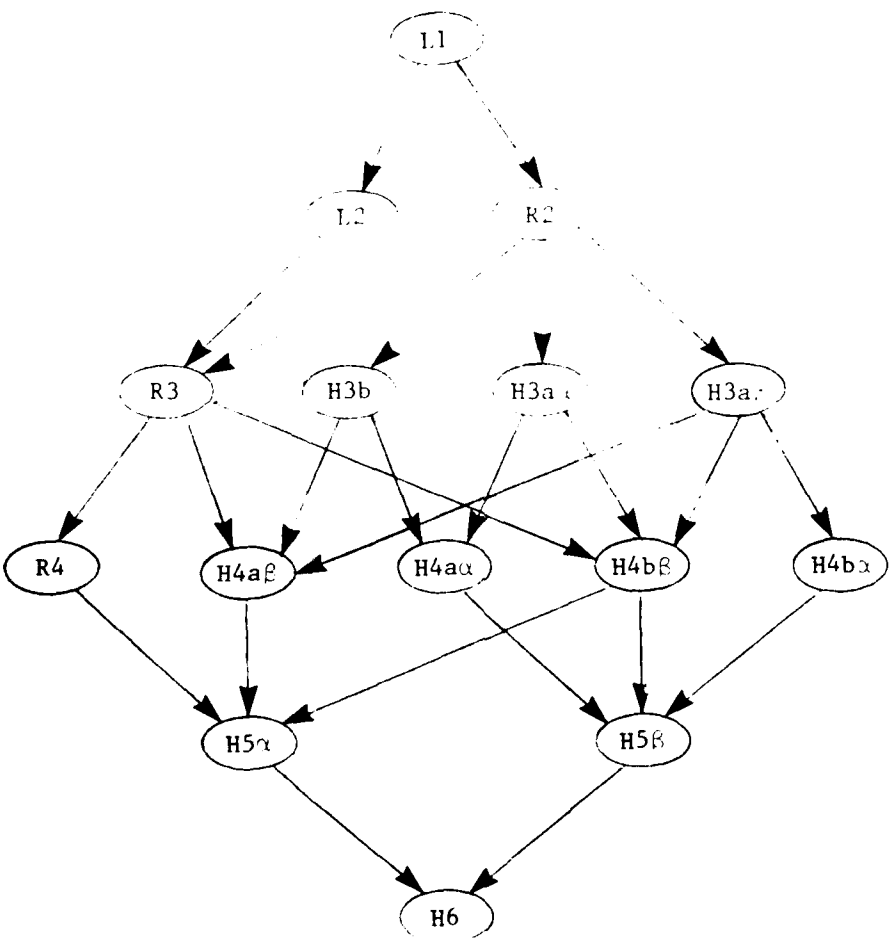


Figure 3-6. Genealogical Chart for Architectures

SECTION 4

CLASSIFICATION OF SPACE-TIME REPRESENTATION

The space-time representation of a completely regular MCN was characterized in the previous section by the dependence matrix D. The hardware configuration was obtained by focusing upon the spatial coordinates of the dependence vectors, which resulted in the truncated dependence matrix D_s. It was observed that the temporal coordinate of all the architectures described in Section 3 was always equal to 1, viz.,

$$D = \begin{bmatrix} & & 1 \\ D_s & & \vdots \\ & & 1 \end{bmatrix} \tag{4.1}$$

so that the dependence matrix D can be easily reconstructed for any given D_s via (4.1). The properties of the corresponding space-time diagram can then be deduced by analysis of the dependence matrix D.

4.1 THE FUNDAMENTAL SPACE-TIME CONFIGURATIONS

Each of the fundamental 11 architectures of Table 3-2 determines a fundamental space-time configuration. We shall focus our attention upon the dependence matrix alone, without considering, for the present, the shape of the boundary surface. Thus, equivalence between the fundamental space-time configurations is established by relating the corresponding dependence matrices by linear transformations. A simple analysis (see Appendix A) shows that every dependence matrix with 2 vectors can be transformed into the equivalent (canonical) form

$$D^{(2)} := \begin{bmatrix} 1 & 0 & \\ 0 & 1 & 0 \end{bmatrix}$$

and every dependence matrix with 3 vectors can be transformed into the equivalent form

$$P^{(3)} \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Consequently, L2 ~ R2 and R3 ~ H3a ~ H3b where the tilde (~) denotes equivalence. For dependence matrices with more than 3 vectors it is convenient to establish first a (nonunique) canonical equivalent, i.e., an equivalent dependence matrix whose first three rows are the identity matrix, viz.,

$$D = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \underline{0} & \underline{0} & \underline{1} \\ & \overline{X} & \end{bmatrix}$$

Some canonical-form equivalents are listed in Table 4-1. The full list of canonical equivalents will be discussed in later sections in conjunction with the specification of boundary surfaces in the three-dimensional space-time continuum.

4.2 ARCHITECTURES WITH LOCAL MEMORY

The preceding analysis was based upon the assumption that processors transmit the results of computations to their immediate neighbors and never store them for further use. However, many applications do involve such storage; this is true, in particular, for adaptive system/parameter identification algorithms that store the identified parameters in fixed location within the array and use the signals that flow through each processor to time-update the locally stored parameters. In this section we consider the architectures obtained by providing each processor with a local memory.

TABLE 4-1. CANONICAL FORM EQUIVALENTS FOR FUNDAMENTAL ARCHITECTURES

L1	L2, R2	R3, H3a, H3b	R4
1 0 0	1 0 0	1 0 0	1 0 0
	0 1 0	0 1 0	0 1 0
		0 0 1	0 0 1
			1 -1 1

H4a	H4b	H5	H6
1 0 0	1 0 0	1 0 0	1 0 0
0 1 0	0 1 0	0 1 0	0 1 0
0 0 1	0 0 1	0 0 1	0 0 1
3/2 -1 1/2	-1/2 1 1/2	-1/2 1 1/2	-1/2 1 1/2
			3/2 -1 1/2

Topologically, local memory means the addition of a self-loop to each processor (Figure 4-1). The direction of each interconnecting link can still be chosen in 3 distinct ways, as explained in Section 3.2, resulting in 11 new architectures (Table 4-2). Two important observations have to be made regarding this table:

(i) The number of dependence vectors is larger by one than the number of interconnections. Thus, for instance, RM3 has 4 direction vectors, not 3.

(ii) The length of the last dependence vector, corresponding to the local memory, equals the temporal displacement between two consecutive occurrences of the same processor in the space-time configuration. Thus, in general, this displacement is 1, except for L2, R3, R4 whose temporal displacement is 2 (corresponding to a periodicity index of 2), and except for H3b whose temporal displacement is 3 (corresponding to a periodicity index of 3).

Local memory can also be used to interleave computations and achieve increased throughput with architectures whose relative throughput without memory is less than 1. This possibility will be discussed in Section 4.4.

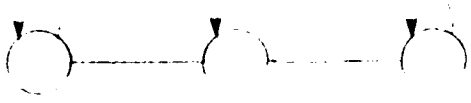
Analysis of equivalence between space-time configurations with local memory reveals that:

(i) LM1, which has 2 linearly independent dependence vectors, is equivalent to L2, R2.

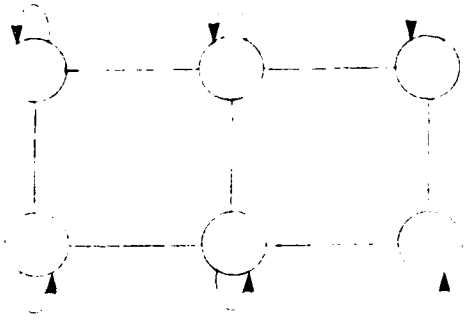
(ii) RM2, which has 3 linearly independent dependence vectors is equivalent to R3, H3a, H3b.

(iii) HM3a, which has 4 dependence vectors, is equivalent to R4.

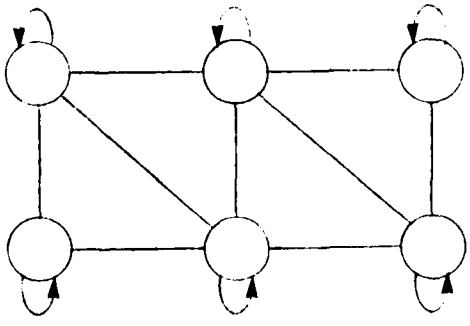
In all three cases we can trade interconnecting links for memory, thereby



a. The Linear Topology with Memory (LM)



b. The Rectangular Topology with Memory (RM)



c. The Hexagonal Topology with Memory (HM)

Figure 4-1. The Three Fundamental Topologies with Local Memory

TABLE 4 2. THE FUNDAMENTAL HARDWARE ARCHITECTURES WITH LOCAL MEMORY

LM1	LM2	RM2	RM3	RM4
1 0 1	1 0 1	1 0 1	1 0 1	1 0 1
0 0 1	-1 0 1	0 1 1	-1 0 1	-1 0 1
	0 0 2	0 0 1	0 1 1	0 1 1
			0 0 2	0 -1 -1
				0 0 2

HM3a	HM3b	MH4a	HM4b	HM5	HM6
1 0 1	1 0 1	1 0 1	1 0 1	1 0 1	1 0 1
0 1 1	0 1 1	0 1 1	0 -1 1	-1 0 1	-1 0 1
1 1 1	-1 -1 1	1 1 1	1 1 1	0 1 1	0 1 1
0 0 1	0 0 3	-1 -1 1	-1 -1 1	0 -1 1	0 -1 1
		0 0 1	0 0 1	1 1 1	1 1 1
				0 0 1	-1 -1 1
					0 0 1

reducing the number of physical wires required to construct a realization of the architecture and simplifying the layout problem for VLSI implementation. Thus, for instance, the R2 architecture which requires a planar network of processors with 4 interconnecting ports at each processor can be replaced by LM1 which requires a linear network of processors with 2 interconnecting ports at each processor and a local memory. Even more remarkably, the same replacement also trades low throughput configurations for high throughput ones.

4.3 BOUNDARY ANALYSIS

The relation between boundary shapes and equivalence between (planar) architectures has been examined in Section 3.4 The combination of topology and boundary has produced 15 distinct architectures which were summarized in Table 3-2. Since each one of these architectures has a rectangular boundary, the resulting space-time configuration always occupies a rectangular prism (with the exception of low-dimensional architectures such as L1, L2, R2 whose space-time configurations occupy 1 or 2-dimensional subspaces).

Since linear transformations change the shape of the boundary, the equivalence between space-time configuration, discussed in Sections 4.1 - 4.2, has to be reexamined to include the effects of boundary transformations. It will be sufficient to carry out this analysis only for collections of space-time configurations which have been declared as equivalent in the preceding sections.

4.3.1 The Configurations LM1, L2, R2

The configurations LM1, R2 can be considered equivalent only when we assume that a single set of inputs is applied to R2 (rather than a time-series). In this case R2 is characterized by

$$\begin{bmatrix} D \\ B \end{bmatrix} = \left[\begin{array}{ccc} 1 & 0 & 1 \\ 0 & 1 & 1 \\ \hline 1 & 0 & 1 \\ 0 & 1 & 1 \end{array}\right]$$

while LM1 is characterized by

$$\begin{bmatrix} D \\ B \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

and the two are equivalent, being related by a linear transformation, viz.,

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

On the other hand, LM1 and R2 are not equivalent to L2 for which

$$\begin{bmatrix} D \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ -1 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

The D-part of this characterization can be related to the D-part of LM1, viz.,

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 \\ * & * & * \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

where the asterisks denote entries which can be chosen arbitrarily (subject to the nonsingularity constraint of the linear transformation). However, when the dependence matrix is combined with the boundary matrix we obtain

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ -1 & 0 & 1 \\ 1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

which does not match the B-part of L2. When the inverse of this transformation is applied to the dependence and boundary matrices of L2, viz.,

$$\begin{bmatrix} 1 & 0 & 1 \\ -1 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1/2 & 0 & 0 \\ 0 & 1 & 0 \\ 1/2 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \\ 1/2 & 0 & 1 \end{bmatrix}$$

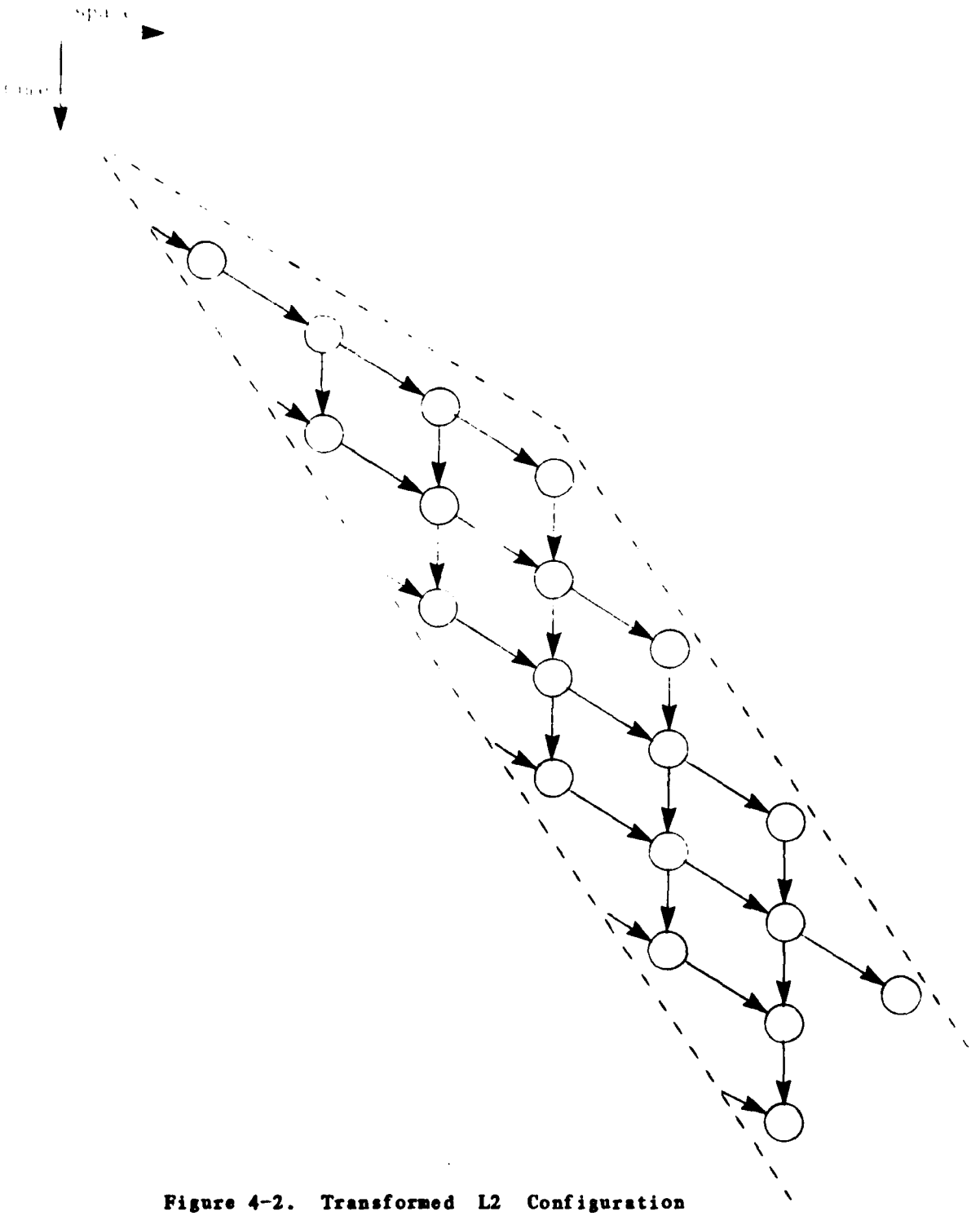


Figure 4-2. Transformed L2 Configuration

the resulting configuration (Figure 4.2) is equivalent to an LM1 configuration of infinite order, the finite active part of the architecture is shifted one cell to the right every time a new input is applied. Thus, in summary, LM1 and R2 are equivalent to each other but not to L2.

4.3.2 The Configurations RM2, R3, H3a, H3b

The truncated boundary matrix of these configurations was chosen in Section 3.4 as

$$B_s = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

namely, the rectangular boundary. The corresponding boundary surface in the space-time continuum is, therefore, characterized by the boundary matrix

$$B = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \qquad (4.1)$$

When this boundary matrix is combined with the dependence matrices of RM2, R3, H3aα, H3aβ, H3b, equivalence is destroyed. For instance, trying to relate H3aα to RM2 we obtain

$$\begin{bmatrix} D \\ B \end{bmatrix}_{H3a\alpha} \cdot \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

The resulting D-part coincides with the dependence matrix of RM2, but the boundary surface is different. The configuration obtained above by transforming H3aα is in fact an RM2 hardware of infinite order in which a finite active segment shifts along the diagonal by one cell each time a set of inputs is applied to the array. This is, in fact, precisely what happens in systolic arrays for matrix multiplication. The configuration H3aα (of Weiser and Davis [12]) is suited for multiplying banded matrices. When the same problem is implemented on an RM2 configuration (of S.Y. Kung [13]) most cells in the array are idle while a small active rectangle,

corresponding to the bandwidth of the given matrices, shifts along the main diagonal of the array. In analogy, while multiplying two matrices with no structure is carried out efficiently by an RM2 array, solving the same problem on an H3a configuration involves many idle cells and a small active segment that shifts along the main diagonal.

4.3.3 The Configurations HM3a, R4

These configurations have the same boundary matrix, given by (4.1), as RM2, R3, H3a and H3b. Since their dependence matrices are different, we conclude that HM3aα, HM3aβ, R4 are distinct configurations when the shape of boundary surface is taken into account.

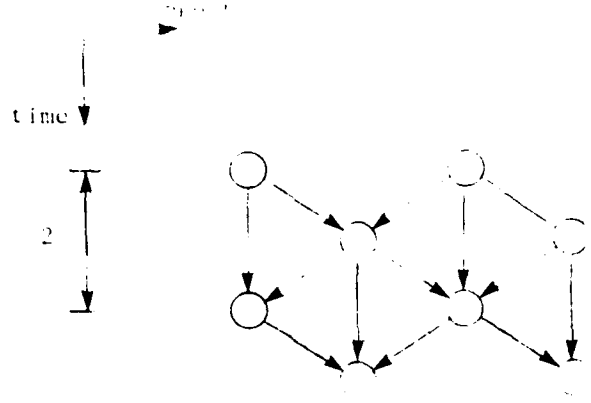
4.3.4 Summary

When boundary considerations are taken into account each of the 15 architectures of Table 3-2 is distinct and cannot be related by equivalence to any other architecture in this table. Incorporating local memory results in doubling the total number of distinct configurations to 30.

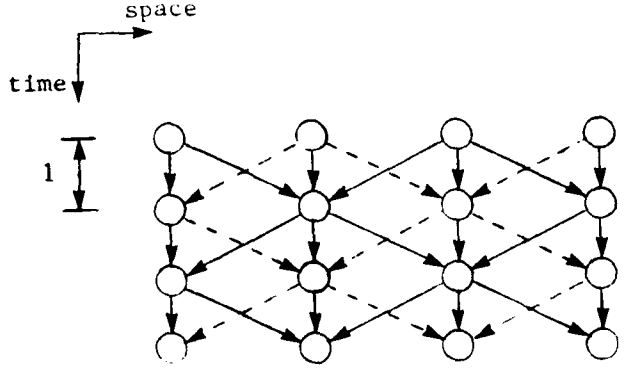
4.4 INTERLEAVING ARCHITECTURES BY LOCAL MEMORY

The introduction of local memory in Section 4.2 involved the assumption that locally stored data remain in memory until required, which makes particular sense in data driven realization. Consequently, the duration of storage for some architectures (L2, R3, R4, H3b) was longer than one time unit. This fact can be used to construct new architectures with higher throughput, by interleaving computations in time and connecting the interleaved computations via the local memory.

The simplest example of such construction is the architecture L2. Without interleaving the throughput of L2, LM2 is 1/2 (Figure 4-3a). With interleaving, which involves superimposing in time two L2 schemes and interconnecting them via local memory, the resulting LiM2 configuration



a. Without Interleaving (LM2)



b. With Interleaving (LiM2)

Figure 4-3. Interleaving via Local Memory

(Figure 4-3b) has throughput = 1. A similar approach produces the architectures RiM3, RiM4 and HiM3b, whose characterizations are given in Table 4-3. The difference between these architectures and their noninterleaved counterparts is the shortening of the local memory dependence vector from either [0 0 2] or [0 0 3] to [0 0 1].

TABLE 4-3. DEPENDENCE MATRICES FOR INTERLEAVED ARCHITECTURES

LiM2	RiM3	RiM4	HiM3b
1 0 1	1 0 1	1 0 1	1 0 1
-1 0 1	-1 0 1	-1 0 1	0 1 1
0 0 1	0 1 1	0 1 1	-1 -1 1
	0 0 1	0 -1 1	0 0 1
		0 0 1	

4.5 SUMMARY

Space-time configurations have been classified by topology, interconnection pattern, shape of boundary, existence of local memory and interleaving. The 15 fundamental architectures of Table 3-2 give rise to another 15 configurations involving local memory. These, in turn, give rise to 4 interleaved configurations, producing a total of 34 distinct space-time configurations.

Ignoring the shape of the boundary surface results in 20 distinct configurations:

1)	L1	11)	H4aα, H4aβ
2)	LM1, L2, R2	12)	H4bα, H4bβ
3)	LM2	13)	RM4
4)	LiM2	14)	RiM4
5)	RM2, R3, H3aα, H3aβ, H3b	15)	HM4aα, HM4aβ
6)	RM3	16)	HM4bα, HM4bβ
7)	RiM3	17)	H5α, H5β
8)	HM3aα, HM3aβ, R4	18)	HM5α, HM5β
9)	HM3b	19)	H6
10)	HiM3b	20)	HM6

Ignoring, in addition, the details of local memory (and, consequently, of interleaving) results in 8 distinct configurations only as in Table 4-1.

Choosing the optimal configuration for a given computational scheme requires a specification of both the interconnection pattern and the boundary shape. This can be accomplished only when specific details of the corresponding computational scheme are taken into account (e.g., bandedness of matrices to be multiplied). When only partial information is considered the designer is often able to choose the interconnection pattern but not the boundary. Thus, multiplication of two matrices can be implemented in any of the five equivalent hardware configurations RM2 [13], R3 [14], H3aα [12], H3aβ, H3b [11]. However, RM2 will be optimal if both matrices have no particular structure; R3 will be optimal if only one of the matrices is banded; and H3aα (or H3aβ) will be optimal if both matrices are banded. It is an historical curiosity that the first systolic array for matrix multiplication, H3b, is never optimal, because it has relative throughput of 1/3 and is otherwise equivalent to H3a.

SECTION 5
CONCLUSIONS

A classification of canonical realizations for completely regular modular computing networks has been presented. Three levels of abstraction were considered: topology, architecture and space-time representation. The analysis revealed 3 canonical topologies, 15 canonical architectures and 34 canonical space-time configurations. It was shown that the unique canonical counterpart of any given topology, architecture or space-time configuration is obtained via a simple (and unique) transformation of the corresponding dependence and boundary matrices. It was also shown that only rectangular boundaries are required to implement any canonical realization. While ignoring boundary details allows some flexibility of design, it also results in inefficient implementations, as explained in Section 4.5.

It is interesting to observe that only a small fraction of the architectures described in this memo have actually been used in the design of parallel algorithms. The most commonly encountered architectures are the linear ones (L2, L1M) which are used for linear filtering (= convolution, polynomial multiplication) and related computations. Next comes the rectangular architecture RM2 and its equivalents--R3, H3a, H3b--which are used in matrix products, matrix triangularizations, solutions of linear equations, QR-factorizations for eigenvalue problems, and adaptive multichannel least-squares algorithms. Thus, all applications involved, to date, are only architectures with 3 dependence vectors or less. Notice also that the classical pipeline (L1) has no use as a signal processing architecture.

The concept of completely regular MCNs involves topologies which are mosaics or completely regular graphs (excluding the boundaries). When this requirement is relaxed to allow regular (but not completely regular) planar graphs, a large variety of new architectures becomes feasible, including regular trees, self-similar graphs (corresponding to self-recursive algorithms) and triangular mosaics. Such configurations, which occur in

various optimization and searching problems, seem to have few signal processing applications.

REFERENCES

[1] J.P. Roth and L.S. Levy, 'Equivalence of Hardware and Software,' Research Report RC 9464, IBM Watson Center, Yorktown Heights, NY, May 1982.

[2] M.C. Chen and C.A. Mead, 'Concurrent Algorithms as Space-Time Recursion Equations,' in S.Y. Kung, et al. (eds.), Modern Signal Processing and VLSI, Prentice Hall, 1984.

[3] W.L. Miranker and A. Winkler, 'Spacetime Representations of Systolic Computational Structures,' IBM Research Report RC 9775, Dec. 1982.

[4] P.R. Cappello and K. Steiglitz, 'Unifying VLSI Array Design with Linear Transformations in Space-Time,' Technical Report TRCS 83-03, University of California, Santa Barbara, Dec. 1983.

[5] D.I. Moldovan, 'On the Design of Algorithms for VLSI Systolic Arrays,' Proceedings of the IEEE, Vol. 71, Jan. 1983, pp. 113-120.

[6] P. Quinton, 'The Systematic Design of Systolic Arrays,' IRISA Report No. 193, France, Apr. 1983.

[7] H. Lev-Ari, 'Modular Computing Networks: A New Methodology for Analysis and Design of Parallel Algorithms/Architectures,' ISI Technical Memo, ISI-29, Dec. 1983.

[8] B. Lisper, 'Description and Synthesis of Systolic Arrays,' The Royal Institute of Technology Report TRITA-NA-8318, Stockholm, Sweden, 1983.

[9] C.J. Kuo, B.C. Levy, B.R. Musicus, 'The Specification and Verification of Systolic Wave Algorithms,' MIT Report LIDS-P-1368, Cambridge, MA, March 1984.

[10] R.M. Karp, R.E. Miller, and S. Winograd, 'The Organization of Computations for Uniform Recurrence Equations,' JACM, Vol. 14, July 1967, pp. 563-590.

[11] H.T. Kung, 'Why Systolic Architectures?' IEEE Computer, pp. 37-46, January 1982.

[12] U. Weiser and A. Davis, 'A Wavefront Notation Tool for VLSI Array Design,' in H.T. Kung, et al. (eds.), VLSI Systems and Computations, Computer Science Press, 1981.

[13] S.Y. Kung, et al., 'Wavefront Array Processor: Language, Architecture and Applications,' IEEE Trans. Comp., Vol. C-31, pp. 1054-1066, Nov. 1982.

[14] S.K. Rao and T. Kailath, 'Digital Filtering in VLSI,' ISL Technical Report, Dept. of EE, Stanford University, Jan. 1984.

[15] O. Ore, Graphs and Their Uses, Math. Assoc. of America, Yale, 1963.

APPENDIX A

EQUIVALENCE VIA LINEAR TRANSFORMATIONS

Two dependence matrices, say, D_1, D_2, are considered equivalent when there exists a nonsingular linear transformation T and a permutation matrix P such that

$$D_2 = PD_1T \tag{A.1}$$

This relation is clearly reflexive (with $P = I$, $T = I$), symmetric and transitive, so 'equivalence' is indeed an equivalence-type relation.

Denoting the length of dependence vectors by n, and the number of dependence vectors by p, we conclude that every dependence matrix with $p \leq n$ and full (row) rank is equivalent to

$$D^{(p)} := [I_p \quad 0] \tag{A.2}$$

which will be defined as the canonical equivalent of such dependence matrices. When $p > n$, and the dependence matrix has full (column) rank, we can always find a permutation matrix P so that

$$PD = \begin{bmatrix} I \\ X \end{bmatrix} T \tag{A.3}$$

where T consists of the first n rows of the permuted matrix PD. Thus, the canonical equivalent of dependence matrices with $p > n$ is of the form (A.3) and the properties of D can be studied by examining the structure of the smaller matrix X.

However, since the submatrix X in (A.3) is not unique, it is required first to find all possible canonical equivalents to a given dependence matrix D. This can be done by applying all possible $p!$ permutations P to the rows of D and then computing X via (A.3). However, not all

permutations generate distinct X matrices. In particular, if we apply a permutation of the form

$$P = \begin{bmatrix} P_1 & 0 \\ 0 & P_2 \end{bmatrix} \qquad (A.4)$$

where P_1, P_2 are permutation matrices of sizes nxn and (p-n) x (p-n), respectively, the resulting canonical equivalent is obtained by solving the equation

$$PD = \begin{bmatrix} P_1 & 0 \\ 0 & P_2 \end{bmatrix} \begin{bmatrix} D_1 \\ D_2 \end{bmatrix} = \begin{bmatrix} I \\ X \end{bmatrix} T$$

which implies that $T = P_1 D_1$ and (assuming D_1 is nonsingular)

$$X = P_2 D_2 D_1^{-1} P_1^{-1}$$

Thus, X is simply some row and column permutation of the fundamental solution $D_2 D_1^{-1}$. To obtain X-matrices that are not permutations of the fundamental solution it is necessary to choose a permutation matrix P which does not have the block diagonal form (A.4). There are only $\binom{p}{n}$ ways of doing so, which is much less than (p!). Moreover, not all of these choices result in distinct X-blocks. Thus, for instance, the dependence matrix

$$D = \begin{bmatrix} 1 & 0 & 1 \\ -1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & -1 & 1 \end{bmatrix}$$

has only one canonical equivalent, viz.,

$$D \sim \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & -1 & 1 \end{bmatrix}$$

while there are in general 24 possible permutations of its rows and 4 ways to choose these permutations in a form that differs from (A.4).

In summary, once all possible canonical equivalents of a given D have been computed it is relatively easy to test whether some other dependence matrix $\hat{D}$ is equivalent to D. One only needs to compute a single canonical equivalent of $\hat{D}$ and compare it to the collection of canonical equivalents of D: a match indicates that $\hat{D}$ is indeed equivalent to D.


DATE:
ILME